IMPLEMENTATION AND ANALYSIS OF PACKET PRIORITY QUEUE IN WIRELESS ACCESS FOR THE VEHICULAR ENVIRONMENT

Submitted to

The Engineering Honors Committee

119 Hitchcock Hall

College of Engineering

The Ohio State University

Columbus, Ohio 43210

By

Allen Chih Jen Chang

241 Sunset Cove

Columbus, Ohio 43202

May 22, 2009

ABSTRACT

This document proposes methods of transmission with different priority level in the Wireless Access for the Vehicular Environment (WAVE) by a simulated slower system with 802.11p protocol. In the vehicular environment, data priority could be the most important issue for safety. There are four different priority levels in the Service Channel (SCH), Control Channel (CCH) and Car-to-Car Channel (C2C) before transmission, which are background, best effort, video and voice in increasing priority level. From [1], a queue data structure would be needed to separate those four different priority level packets at the MAC layer. By implementing a slower system with both On-Broad units (OBU) and Road-Side Units (RSU) in both python and C, and also by using different Arbitration Inter-Frame Space (AIFS) and adjustable Contention Window (CW) schemes, the communication in between machines will then have more effectiveness in safety.

ACKNOWLEDGMENTS

I believe that the reason I am where I am today, is due to those who had been there for me through the most difficult times in my life, those who have encouraged me to become better and provided me with enough push for me to carry on.

In 2003, my family won the permanent resident lottery hold yearly by the U.S. government. It was with that less than one percent of possibility that has made a dramatic change within my entire family. At that time, I was a senior student in high school with no particular ultimate goal for future studies. I want to thank my parents who have supported me for studying in the U.S. But in Particular, I want to thank my grandfather and grandmother who had passed away in Taipei in 2007 while I was sleeping in my apartment in Columbus. My grandparents were refugees of the Chinese Civil War in 1949. They started out with nothing but with hard work and determination of a better life for their descendents, they have managed to overcome poverty and were able to provide us with enough financial capacity so that my sisters and I have sufficient funding for our educations in the U.S. Without my family's support, I would not have the opportunity to finish my undergraduate degree with distinction.

I would like to thank Dr. Ekici, Dr. Potter and PHD candidate Scott Biddlestone at the Ohio State University. I thank Dr. Ekici for giving me a chance to work on WAVE project, and thank him for listening and helping me during the weekly meetings and his office hours. I would also like to thank Dr. Potter for the wonderful group project class in wireless communication, which has been a great inspiration to me. And also, he always helps me during his break time. Finally, I would like to thank Scott, my sole mentor in

WAVE project, who always tries to help me better understand the program which is more complicated than spaghetti.

I also thank everyone who has helped me with life in general.

VITA

December, 1985	.....	Born – Taipei, Taiwan
2004-2005	.....	Major in Physical Science, St. John's University, Jamaica, NY
2005-	.....	Major in Electrical and Computer Engineering and Minor in Computer Information Science, The Ohio State University, OH
June, 2009 (Projected)	.....	B.S. Electrical and Computer Engineering with Distinction and Minor in Computer Information Science, The Ohio State University

FIELDS OF STUDY

Major Field: Electrical and computer Engineering- Electrical Engineering Specialization

Studies in:

Data Networks, Vehicular Communication Systems	Prof. Eylem Ekici
Digital Signal Processing, Communication Theory	Prof. Lee Potter

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

CHAPTER 1

OVERVIEW

1.1 Introduction

Automobiles have become important part in the modern lifestyle. Communication between automobiles and road side infrastructure has also become an important topic in the special communication environment. “Inter-Vehicle Communication (IVC) systems” with an advanced protocol “802.11p” or so-called “Wireless Access in the Vehicular Environment (WAVE)” have been proposed to increase traveler safety, reduce fuel consumption and pollution, and to maintain connectivity among vehicles and from vehicles to infrastructure. In this project, two different dedicated short-range communications (DSRC) networks are alternatively involving on the channel: Vehicle-to-Vehicle (V2V) networks and Vehicle-to-Infrastructure (V2I) networks. The major difference between 802.11b and 802.11p is that 802.11p does not have RTS-CTS mechanism for collision free condition. In other words, collision will become the most crucial issue on the channel.

Within V2V and V2I networks, there are three different kinds of channels; and they are Control Channel (CCH), Service Channel (SCH) and Car-to-Car Channel (C2C). CCH is a radio channel between OBU and RSU used for exchange of management frames and WAVE short messages (WSMs). SCH is the secondary channels between OBU and RSU used for application specific information exchanges. C2C is the channel

between OBUs used for exchanging safety messages. Both OBUs and RSUs would monitor CCH until a broadcasted WAVE service advertisement (WSA) is received and announces a service that utilizes a SCH. This time period is known as Control Channel intervals (CCH intervals). If OBU receives the WSA and confirms during CCH interval, it will retune to SCH until the next CCH period; otherwise OBU will retune back to C2C for safety messages among the OBUs.

For safety issues, packets have their own priority level to prevent sudden emergences. To clarify, specification has already specified four different packets with different information contained in both V2V and V2I networks. However, among those four different packets, they have also defined different priority levels. [1] suggests a queue data structure in MAC layer which stores packets. Furthermore, among several identical systems, higher priority packets should have more opportunity to obtain the channel and to be transmitted. The mechanisms to be implemented on the channel are called Arbitration Inter-Frame Space (AIFS) and adjustable Contention Window (CW). Arbitration Inter-Frame Space is an offset that depends on various priorities, and adjustable Contention Window is used to lower the collision rate and waiting period.

In this project, a similar system developed using GNU Radio and Universal Software Radio Peripheral (USRP) with two OBUs and one RSU device both software implementation and analysis of those mechanisms will be presented in special cases such as high traffic, low traffic, retries, cross affection of traffic and retries, code validation and actual transmission time.

1.2 Objectives

This project is a research lead by Dr. Ekici at the Ohio State University. The overall objective for Inter-Vehicle Communication (IVC) systems is to support convenience applications, including personal communications, mobile office, location based information, car related mobility services, live video streaming, and Internet access. The objective of the implementation and analysis of packet priority queue is to make transmission of different priority packets efficiently and orderly in the environment which lots of different packets are waiting to be transmitted. By fulfilling this objective, a special queue structure coded in Python along with contention window adjustment function in MAC layer will be presented and tested. After completing the pre-queue system, packets will be separated into groups with its priority levels. By functioning with the groups of queues, high priority packets will have relatively high possibility to obtain the medium. Likewise, relatively low priority packets will have low possibility to obtain the medium. As the result, the system can determine which packet to be sent depending on their priority level on the channel with other system in communicable range. Some observations will be distributed and analyzed to support the thesis. The observations include high traffic, low traffic, retries, cross affection of traffic and retries, code validation and actual transmission time.

1.3 System Diagram

From Section 1.1, we already know that there are two different units, on-board units and road-side units in the system. And, each unit has to switch among three different channels: Control Channel, Service Channel and Car-to-Car Channel.

1.3.1 GNU Radio and the Universal Software Radio Peripheral

GNU Radio, a software defined radio package designed to implement common radio functions, is being used in this entire project. Without GNU Radio package, the Universal Software Radio Peripheral is the hardware being used to support GNU Radio which cannot be used without a hardware radio system. Since GNU Radio package does have multiple mixers, filters, amplifiers, modulators and demodulators implemented as blocks, it's easier for us to implement our special case system. The USRP system is being used as the hardware for both transmitter and receiver. In this thesis, the most focusing part is Network Allocation Vector (NAV) section in MAC layer in GNU Radio package.

1.3.2 Channel Switching

There are three different kinds of channels: Control Channel (CCH), Service Channel (SCH) and Car-to-Car Channel (C2C), and they are alternating one by one with conditions.

Basically, all units have to be synchronized and start at one point. First, each machine should monitor on CCH. If OBU receives WAVE short message protocol (WSMP) messages from RSU, then OBU can determine whether the service is wanted. If it is, then go to SCH for service; otherwise, go back to C2C for car safety. In between each channel alternation, there is a time period called "guard interval". Guard interval is used as a tolerance period when units are switching channel.

Figure (1.1) demonstrates the scenario of channel alternation. During the first cycle, OBU receives a packet from RSU called WSMP#1 and makes the decision of no transmission. Then OBU retunes back to C2C when CCH time is up. During the second cycle, OBU receives another packet from RSU called WSMP#2 and makes the decision

of transmission. Then OBU retunes to SCH for service. During the third cycle, OBU receives nothing from RSU. As the result, OBU retunes back to C2C when CCH time is up.

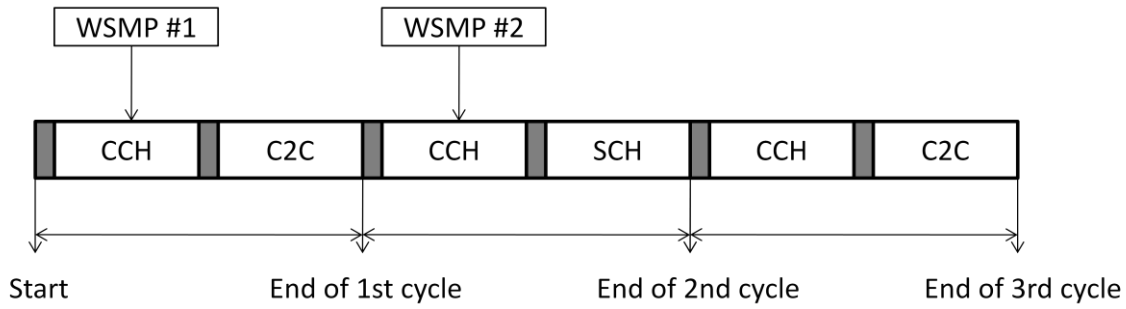


Figure (1.1), example of OBU alternative among CCH, SCH and C2C by conditions

1.3.3 On-Board Unit (OBU)

In this section, a closer look at On-Board Unit including system flowchart will be presented. When OBU is powered up, two main conditions will be alternated: CCH and SCH.

1.3.3.1 Control Channel for OBU

The task in CCH in OBU system is to obtain WSMP messages from RSU, and determine if the unit wants the service. According to [2], if OBU receives something from RSU and decides to make connection, OBU will send a Service Request (SRQ) message back to RSU and retune to SCH for transmission. Otherwise, SCH will be idle

for next transmission period and listen to C2C channel. Flowchart 1 in appendix A demonstrates the flowchart of how OBU deals with CCH period.

1.3.3.2 Service Channel for OBU

The task in SCH in OBU system is to obtain service packets from RSU. According to [2], a basic data network system with ACK mechanism is being used. Therefore, if OBU receives some service packets from RSU, OBU will send an ACK message back to RSU and also determine if there are more packets to be sent from RSU by checking the sequence field in the packet. Otherwise, OBU will be ready to retune back to next CCH period. If OBU does not receive anything from RSU, that means SRQ which has been sent in CCH period is lost. OBU will resend SRQ to RSU if RSU is still in CCH period. Flowchart 2 in appendix A demonstrates the flowchart of how OBU deals with SCH period.

1.3.4 Road-Side Unit (RSU)

In this section, Section like 1.3.2 for OBU, we will discuss the basic logic behind RSU in CCH and SCH.

1.3.4.1 Control Channel for RSU

Since RSU is a service style unit, RSU in CCH will provide service to OBU by WSMP messages which are broadcast style messages. According to [2], if OBU receives WSMP messages from RSU, it will send a SRQ messages back to RSU. Right after RSU receives SRQ, RSU will ready to retune to SCH for service. Flowchart 3 in appendix A demonstrates the flowchart of how RSU deals with CCH period.

1.3.4.2 Service Channel for RSU

Like OBU in SCH period, RSU makes connection with OBU by a simple ACK network. First, RSU will send the packet to the specific OBU by its direct address, and RSU will wait for ACK for OBU. If RSU receives ACK from OBU and still has more packets to be sent, then RSU will keep sending the packets and waiting for ACK until SCH time is up. If RSU does not receive ACK from OBU, then RSU will resend the packet to OBU if time is still allowed. Flowchart 4 in appendix A demonstrates the flowchart of how RSU deals with SCH period.

CHAPTER 2

CONTENTION PERIOD

2.1 Arbitration Inter-Frame Space (AIFS)

According to [3], Arbitration Inter-Frame Space is defined as the minimum time interval for a certain kind of packet to wait between the medium is becoming available and the beginning of transmission. Different AIFS values will yield Different abilities to obtain the channel. For a relatively low priority packet, a high value of AIFS value will be expected. For a relatively high priority packet, a low value of AIFS value will be expected. For example, for a certain type of packet, if AIFS value is small, that means the packet has more priority since it does not have to wait too long to obtain the channel; On the other hand, for a certain type of packet which has large AIFS value, that means the packet has relatively less priority since it waits too long and other units may obtain the channel in advance.

2.1.1 Arbitration Inter-Frame Space Number (AIFSN)

However, for lowering the collision rate among packets in the channel, time in the channel will be chopped up into pieces, and each piece of time interval we call it a "time slot". Following this rule, we can define Arbitration Inter-Frame Space Number to be shown in Equation (2.1).

$$\mathbf{AIFSN(integer)} = \frac{\mathbf{AIFS(sec)}}{\mathbf{a\ time\ slot(sec)}} \qquad (\mathbf{2.1})$$

Finally, we have AIFSN to be the minimum number of time slot for a certain kind of packet to wait between the medium is becoming available and the beginning of transmission. According to [1], AIFSN value is defined among those four packets for their different priority level in table (2.1). The result will yield different priority level among the system in the channel.

packet Context	**AIFSN(CCH)**	**AIFSN(SCH)**
Background	9	7
Best Effort	6	3
Video	3	2
Voice	2	2

Table (2.1), AIFSN values for CCH and SCH

2.1.2 Short Inter-Frame Space (SIFS)

For special packets such as ACK, the waiting time is even smaller. Since ACK packets only exist when the connection is made, and for the systems other than connecting systems will be locked by NAV, we can conclude the no signal will cause interference on the channel (ideally). For performance, ACK packet use short inter-frame space (SIFS) that is a required time interval for hardware system to send the packet and the duration is even shorter than AIFSN=1. Figure (2.1) demonstrates the different minimum time

required for ACK, Voice and Video packets to be transmitted if no collision occurs. In the project, we set SIFS time equals to exactly a slot time or one AISFN.

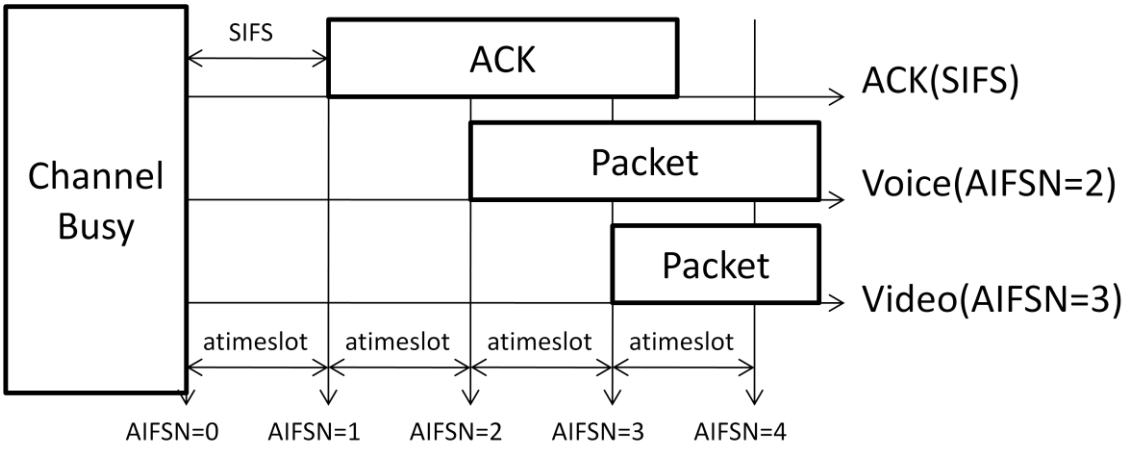


Figure (2.1) the minimum time for ACK, Voice and Video packets to be transmitted

2.1.3 Total Minimum Waiting Time

To conclude, including short inter-frame space before every transmission due to hardware adaption, the total waiting time for different packets will depend on their AIFSN numbers which yield different AIFS value. Finally, the total minimum waiting time will be shown in Equation (2.2).

$$\textbf{minimum wait time} = \textbf{SIFS} + \textbf{AIFSN} * \textbf{atimeslot} \qquad (\mathbf{2.2})$$

Although AIFS value can make different priority levels among different packets by their waiting time, this scheme can only be served as a single connection. To specify, if more than one machines share a channel, it will always cause collision since the minimum waiting time is fixed. Now we have to consider more about the systems in almost collision-free environment.

2.2 Contention Window

According to [3] an [5], Contention Window is defined as an interval from which a random number is drawn to implement the random back-off mechanism. In other word, to prevent collision among the systems in the channel, a random time interval will be served before the packet is going to be sent. For a given range of the random number to be generated, among the packets with the same priority level (AIFSN) on the deck, the smallest contention window value will obtain the medium before other systems. Base on the random number, this scheme can reduce the collision rate in the channel. Figure (2.2) shows the situation that three different systems have three packets with same priority level (AIFSN). In figure (2.2), among those three packets with same priority level, the difference of random contention windows prevents collision in the channel. Finally, system#2 has the lower contention window value and obtains the channel to send the packet.

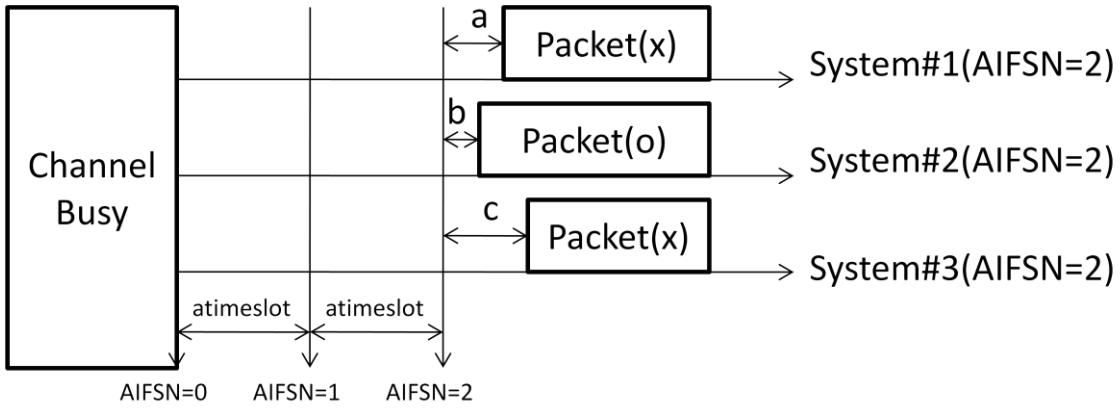


Figure (2.2), three systems with three random contention windows (where b<a<c) and system#2 obtains the channel

2.2.1 Contention Window and Time Slot

Like AISFN in Section 2.1.1, for lowering the collision rate, contention window must matches the synchronized sequence of transmission time. After all, contention window cannot be a random real value but a multiple of a slot time and a random integer value. As the result, contention window in time domain will yield as the following in Equation (2.3):

$$\mathbf{CW = random\ CW\ integer * a\ slot\ time} \qquad (\mathbf{2.3})$$

2.2.2 Time Priority

Although the shortest contention window wins the channel, other systems will have to wait until the next transmission because of Network Allocation Vector (NAV) lock. When the system is searching for the next available medium, the packet that has been waiting for more than one transmission time should have higher priority than others. Since we do not want the other systems to regenerate other random contention window that has the possibility that will be longer than the previous contention window, the new contention window can be the remainder from the previous contention window. As the result, the oldest packet will have greater possibility to be transmitted at next channel available. Figure (2.3) shows the continuous graphic demonstration for figure (2.2).

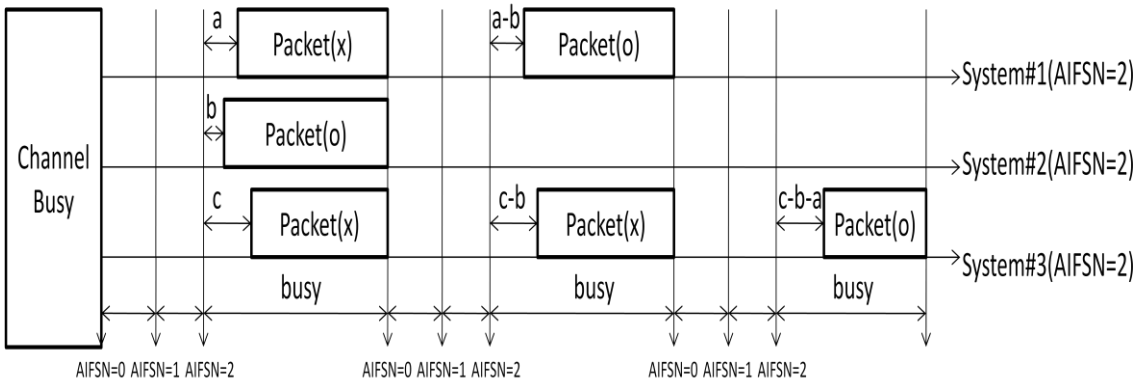


Figure (2.3), three systems with three random contention windows (where b<a<c) and system#2, system#1 and system#3 obtains the channel by order

System#2 will obtain the channel first, and then system#1 and system#3 will reduce their contention window by whenever the systems are being waited. With the same

scheme, system#1 will obtain the channel and then system#3 will finally obtain the channel. In this case, even if some other systems involve into the channel at second idle time, the older the packet is, the greater possibility the packet is going to be sent. Finally, the waiting time for a packet to be transmitted will become as the following in equation (2.4).

$$\textbf{Wait Time} = \textbf{SIFS} + \textbf{AIFS} + \textbf{random integer CW value} * \textbf{a slot time} \quad (\mathbf{2.4})$$

2.2.3 Collisions and Retry

There are two types of collisions, internal collision and external collision. Internal collision is the situation that two or more packets have the same random contention window. In this case, the system is designed to determine the highest priority packet and send it. By time priority in Section 2.2.1, other contention window values will become zero and have relative higher priority than other packet in the same system.

Besides internal collision, external collision will be more critical. External collision occurs when two or more systems obtain the median at the same time and cause interference. There is no efficient method to avoid external collision but the system can avoid another collision by regenerating another longer contention window to lower the possibility of collision.

In case of collision, we increase CW_{min} by 2*(CW+1)-1 every time when the system retries to send the packet until new CW_{min} value is equal to CW_{max} value. Here, we initialize our CW_{min} when the packet obtains the channel.

Finally, total waiting time will be shown in Equation (2.5) below:

$$\textbf{Wait Time} = \textbf{SIFS} + \textbf{AIFS} + \textbf{min}\{\textbf{rand}(\textbf{0}, \textbf{CW}_{\textbf{min}}(\textbf{retry})), \textbf{CW}_{\textbf{max}}\} * \textbf{timeslot} \quad (\textbf{2.5})$$

2.2.4 Actual Contention Window value

According [1], contention window values are various for different priority queues. Also, it does indicate that the maximum retry time is seven. The followings are the actual values listed in tables.

Table (2.2) demonstrates the different contention window value range defined in [1]. From 20.4.4 in [4], we obtain aCW_{min} and aCW_{max} value, which are 15 and 1023. aCW_{min} is defined as the start value of contention windows, and aCW_{max} is defined as the maximum contention windows when collision occurs.

Table (2.3) demonstrates the calculated contention window with retry number. To clarify, we follow the calculus in Section 2.2.3 with its retry time. Some contention window values are stopped at their maximum aCW_{max} value.

Packet Context	**CW_{min}**	**actual**	**CW_{max}**	**actual**
Background(CCH)	aCW_{min}	15	aCW_{max}	1023
Best Effort(CCH)	$(aCW_{min}+1)/2-1$	7	aCW_{min}	15
Video(CCH)	$(aCW_{min}+1)/4-1$	3	$(aCW_{min}+1)/2-1$	7
Voice(CCH)	$(aCW_{min}+1)/4-1$	3	$(aCW_{min}+1)/2-1$	7
Background(SCH)	aCW_{min}	15	aCW_{max}	1023
Best Effort(SCH)	aCW_{min}	15	aCW_{max}	1023
Video(SCH)	$(aCW_{min}+1)/2-1$	7	aCW_{min}	15
Voice(SCH)	$(aCW_{min}+1)/4-1$	3	$(aCW_{min}+1)/2-1$	7

Table (2.2), Contention Window range for SCH and CCH

Packet Context \ retry	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
Background(CCH)	15	31	63	127	255	511	1023	1023
Best Effort(CCH)	7	15	15	15	15	15	15	15
Video(CCH)	3	7	7	7	7	7	7	7
Voice(CCH)	3	7	7	7	7	7	7	7
Background(SCH)	15	31	63	127	255	511	1023	1023
Best Effort(SCH)	15	31	63	127	255	511	1023	1023
Video(SCH)	7	15	15	15	15	15	15	15
Voice(SCH)	3	7	7	7	7	7	7	7

Table (2.3), Contention Window value with retry times for SCH and CCH

2.3 Combination of AIFS and CW

The Combination of Arbitration Inter-Frame Space (AIFS) and Contention Window (CW) yield the priority level among four packets. Figure 4 shows the waiting time among four different packets involving AIFS and CW in CCH and SCH. In figure (2.4), the transmission will randomly occur within the range of contention window.

In the same machine, each of the four queues has its unique AIFS and contention window value for each transmission if no collision occurs. Among several machines, the queue in a certain machine which counts up to sum of their AIFS and CW obtains the channel. The other machines will be locked by NAV until next channel available and transmit with previous CW reminder. If collision occurs, the collided packets will process another back-off mechanism which regenerates another CW value within wider range.

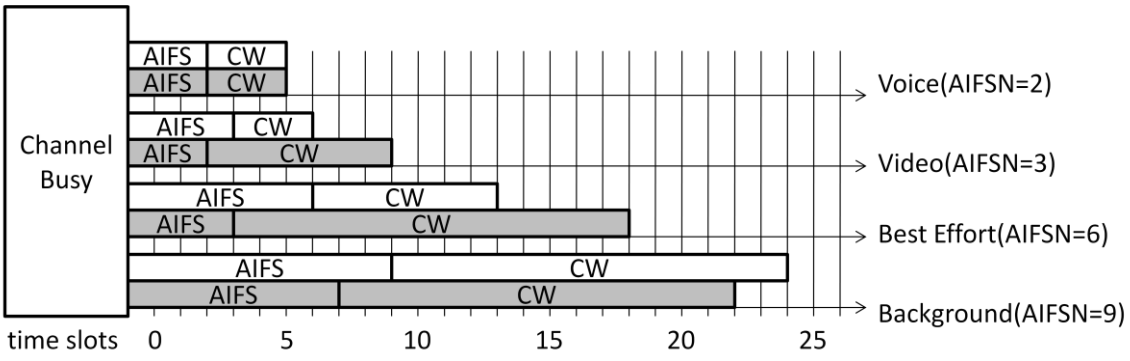


Figure (2.4), total waiting time in CCH (white) and SCH (shaded) among packets

CHAPTER 3

IMPLEMENTATION

3.1 Assumption

According to IEEE [1], the MAC layer should be implemented in four queues with three control blocks. Figure (3.1) shows the basic design of the data structure with four queues, AIFSN/CW counter, internal control block and external control block.

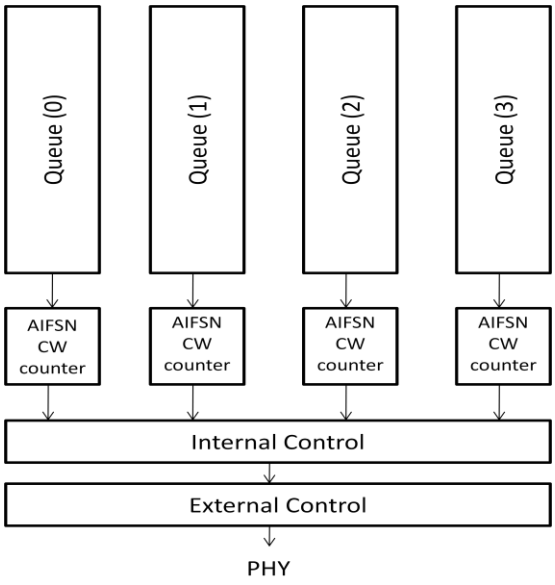


Figure (3.1), data structure in software view with four queues, counters and control blocks

Since CCH and C2C are identical, we can just take care of CCH and SCH, and then have a copy of CCH for C2C. In the system, it is relatively slow and below and requirement in [1]. Another assumption will be the period of a slot time. For convenience, we set the period of a slot time to be identical as SIFS time. This allow us to easily decrement the sum of AIFSN and CW without another state.

For hardware, we have two OBUs and one RSU for testing, so collision part can be ignored. Once we have more devices and more revision version of program, we can add the collision part back and test it.

3.2 Packet Queue

Packet Queue will be used to store packets at MAC layer before they are transmitted. Queue has first-in-first-out (FIFO) scheme, and packets need to be list and aligned by requesting time order. When the system is locked by NAV or idle, the queue starts to enqueue packets if provided from upper layer (Currently, packets are made directly at MAC for testing). Whenever internal control and external control make decision to send the packet, the queue will dequeue the packet and send it to physical layer to be transmitted. To clarify, queue will be the structure for storing the packet before being transmitted.

3.2.1 Structure

Storage for packets will be a big issue for the system. Since the system has maximum memory capacity, a recommended structure can be pointer-style queue.

The basic idea for a pointer-style queue is using pointers to associate packets. Comparing to array-style queue, pointer-style is more complicated with allocating

memory space, but the only advantage will be the memory capacity. Pointer-style queue will allocate a memory space when there is a packet, and array-style queue will allocate all the memory space at beginning. In other words, Pointer-style queue is dynamic and array-style queue is static. In WAVE system, a dynamic data structure will be most ideal than a static data structure. Since the system is processes with lots of conditions, a dynamic structure will give the system a tolerant space of memory, and which is much safer on the road.

3.2.2 Class Description

For four queues in three channels with control units, a class with all those fields can gather everything we want. The largest class called "Queues In Three Channels" which contains the queues in three channels. Each channel will have a class called "Four Queues" which contains four queues and control fields including: a 1X4 array of Current AIFSN+CW value for four queues, a 1X4 array of AIFSN value for four queues, a 1X4 array of initial CW value for four queues, a 1X4 array of maximum CW value for four queues and finally a counter which contains current slot time that has passed during contention period to adjust CW.

As Illustrated in Section 3.2.1, a pointer-style queue is going to be constructed. Each queue has three fields: "start", "end" and "length". Both "start" and "end" are pointers to a class called "packet cell" which contains a packet and pointer called "next". Once a certain packet appears, system will assign the packet into packet field in packet cell, and "start" pointer will point to that packet cell. Also, length of the queue will increment as well. Additionally, all the packets will be put in to packet cells and be linked by points. Figure (3.2) shows and entire class picture for priority queues in three channels.

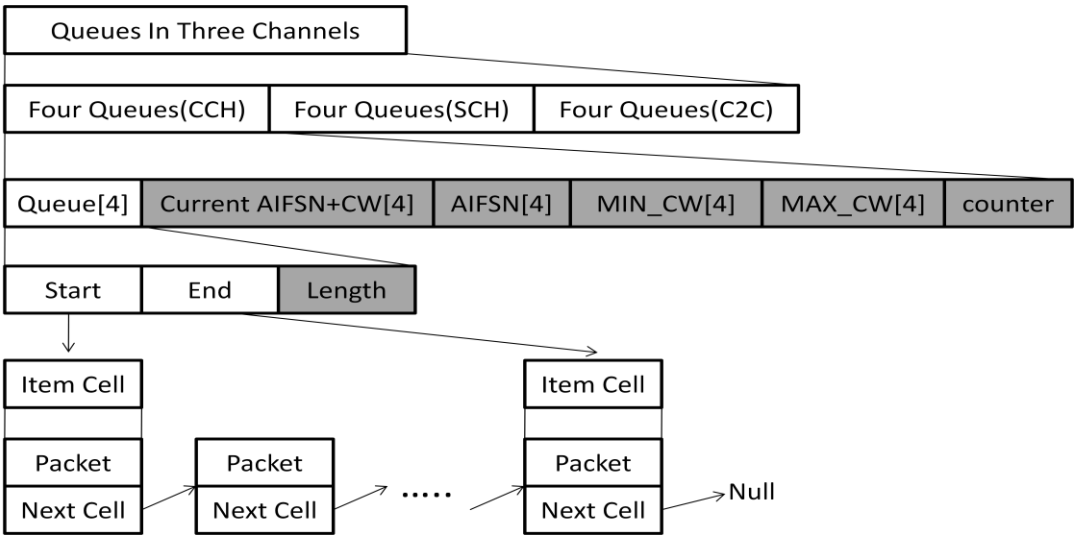


Figure (3.2), The derivation of Class of Queues in Three Channels. Arrows are pointers, and integer value fields are shaded.

3.3 AIFSN/CW Counter

Whenever the medium is idle, or the system is unlocked by NAV technically, AIFSN/CW counters will start counting up from zero to the fewest value of sum of AIFSN and CW values which is in current AIFSN+CW field in Section 3.2.2. Before the medium is idle, each packet on the deck of each queue will be assigned a value which is the result of sum of AIFSN and random CW, and store the value to its corresponding current AIFSN+CW field. AIFSN value is fixed for four queues. CW value can be obtained by random generator with value range. All four queues will start counting down by checking the channel at each beginning of a time slot after the channel is idle for SIFS amount of time. If the channel is idle at the beginning of a certain time slot, each current AIFSN+CW field will have to decrease the value until the counter becomes zero. On top of that, counter has to add up the number of time slots have passed until some current AIFSN+CW field become zero.

Whenever the current AIFSN+CW field becomes zero, it indicates that the corresponding queue obtains the channel. Other queues in the system will adjust their CW value to the next channel available. According to Time priority in Section 2.2.2, the queue which does not obtain the medium will adjust its CW but not AIFSN. The counter solves the queue adjustment. If counter value is greater than its AIFSN, it means CW period has been used in the contention period. Then we just simply add its AIFSN value to the old current AIFNS+CW field for next channel available. On the other hand, if counter value if less than its AIFSN, it means CW period has not been used in the contention period. Logically, current AIFSN+CW field will have to reset to the previous value, and which is exactly counter + previous current AIFSN+CW field.

Not only the packet from own system obtains the medium will cause to adjust other AIFSN+CW field in the same system, but also the packet from other system obtains the medium will cause to adjust all the AIFSN+CW field in other systems. Also, beyond and include second try due to collision, the random generator will regenerate CW number with wider range calculate. Flowchart 5 in appendix A shows the flowchart of the procedure of each AIFSN/CW counter.

3.4 Internal control and External control

Internal and external control blocks make decision when internal collision and channel collision occur. Flowchart 5 in appendix A, there are four different cases depending next state:

3.4.1 Case (1): Channel is idle and AIFS+CW is not zero

When a certain current AIFS+CW is not equal to zero, it indicates that the corresponding packet has yet reached the total waiting time which is the sum of AIFSN and CW values. Since there is no internal and external collision, the system state goes back to decrement value of current AIFS+CW field and increment the counter.

3.4.2 Case (2): Channel is busy and AIFS+CW is not zero

When a certain current AIFS+CW is not equal to zero, it indicates that the corresponding packet has yet reached the total waiting time which is the sum of AIFSN and CW values. Here assuming some packets obtain the channel, and then we adjust the current value in current AIFS+CW field and go back to channel busy for other transmission. NAV is either lock or unlock depends on the transmission system.

3.4.3 Case (3): Internal collision and AIFS+CW is zero

When a certain current AIFS+CW is equal to zero, it indicates that the corresponding packet has reached the total waiting time which is the sum of AIFSN and CW values and is ready to be transmitted. Here assuming some packets in the same system or in different systems also obtain the channel, we can conclude that internal or external collision occurs. If internal collision occurs, we make transmission decision among packets by priority level; for low priority packet, we adjust the current AIFS+CW value along with other queues by the counter and go back to channel busy for other transmission. If external collision occurs, we make back-off on the collided packets and regenerate and save the new CW value in AIFS+CW and go back to channel idle. If transmission occurs, NAV is either lock or unlock depends on the transmission system.

3.4.4 Case (4): No collision and AIFS+CW is zero

When a certain current AIFS+CW is equal to zero, it indicates that the corresponding packet has reached the total waiting time which is the sum of AIFSN and CW values and is ready to be transmitted. Here assuming that no packet in the same system or in different systems also obtains the channel, we can conclude that no internal or external collision occurs. To start transmission, a duplicated packet from the queue will be sent to physical layer for transmission. After the system receives ACK from specific system, system will dequeue the packet from the queue. NAV is either lock or unlock depends on the transmission system.

CHAPTER 4

DATA ANALYSIS

From Section 1.1, collision can be the most crucial problem for the system without RTS-CTS mechanism. In this chapter, collision analysis will be the most important topic. Since CW values among different queues are exactly random number, so the only way we can analyze without simulating the system is to calculate the probability of collision. Some special case tests and observations will be made after the completion of the entire Inter-Vehicle Communication (IVC) systems to support the thesis. Special cases include high traffic, low traffic, retries, and cross affection of traffic and retries.

4.1 Probability of Collision in High-Low Traffic Conditions

The definition of high-low traffic depends on the number of systems in the communicable range. For a specific packet with its CW value, the number combination of non-collision among the entire range will be as equation (4.1). And the total number of combination of CW value among the entire will be as equation (4.2). Finally, the probability of collision will be one subtracts the total probability of non-collision condition as equation (4.3).

$$\frac{(\mathbf{CW}+\mathbf{1})!}{((\mathbf{CW}+\mathbf{1})-\mathbf{systems})!} \qquad (\mathbf{4.1})$$

$$(\mathbf{CW}+\mathbf{1})^{\mathbf{systems}} \qquad (\mathbf{4.2})$$

$$\mathbf{1} - \left(\frac{(\,\mathbf{CW} + \mathbf{1}\,)!}{(\,(\mathbf{CW} + \mathbf{1}) - \mathbf{systems})!\,(\mathbf{CW} + \mathbf{1})^{\mathbf{systems}}} \right) \qquad (\mathbf{4.3})$$

Figure (4.1) and figure (4.2) shows the probability of collision with various numbers of systems in communicable range in CCH and SCH. As the graph goes right, the number of systems in the range is increasing, and the probability of collision goes high because more and more systems are trying to obtain the medium with relatively tiny CW value. On the other hand, as the graph goes left, the number of systems id decreasing, and the probability of collision goes low because fewer systems are trying to obtain the medium with relatively large CW value. For example, in the up-left graph in figure (4.1) and figure (4.2) shows the probability of collision with various systems in range in CCH and SCH for 1st try. For the highest priority voice packets, if more than four systems try to join the contention period, collision will be guaranteed. Since voice packets have a tiny CW value of three, which means they have zero, one, two and three of its possible values. If there are five systems in the range, it's impossible to fit those five CW values into four possible values. So it's 100% collision. Otherwise, if there are only two systems in the range, the probability of collision will be 25%.

To conclude, a relative high traffic environment will cause the probability of collision increase; a relative low traffic environment will cause the probability of collision decrease.

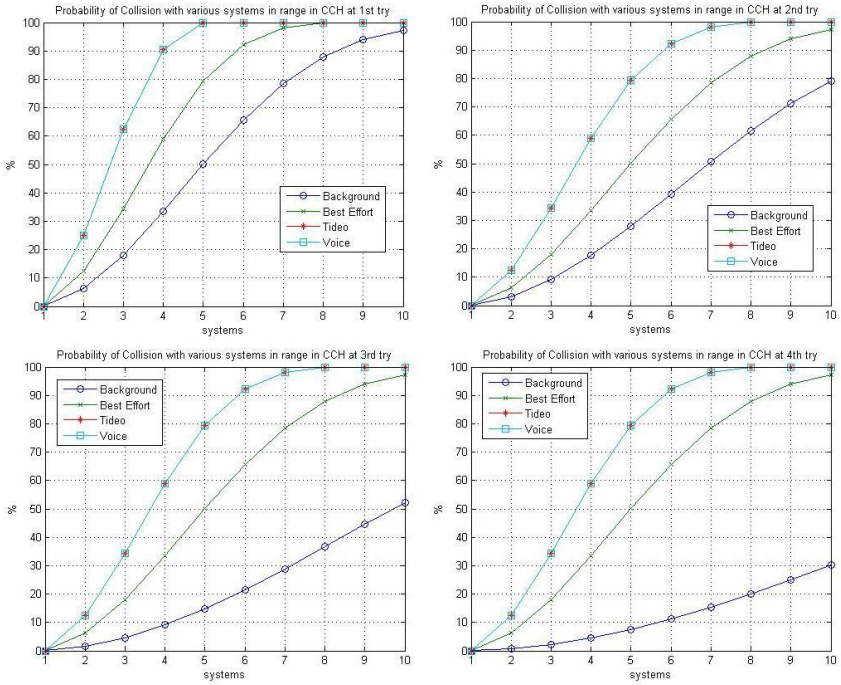


Figure (4.1), Probability of Collision with various systems for 1st, 2nd, 3rd and 4th try for CCH

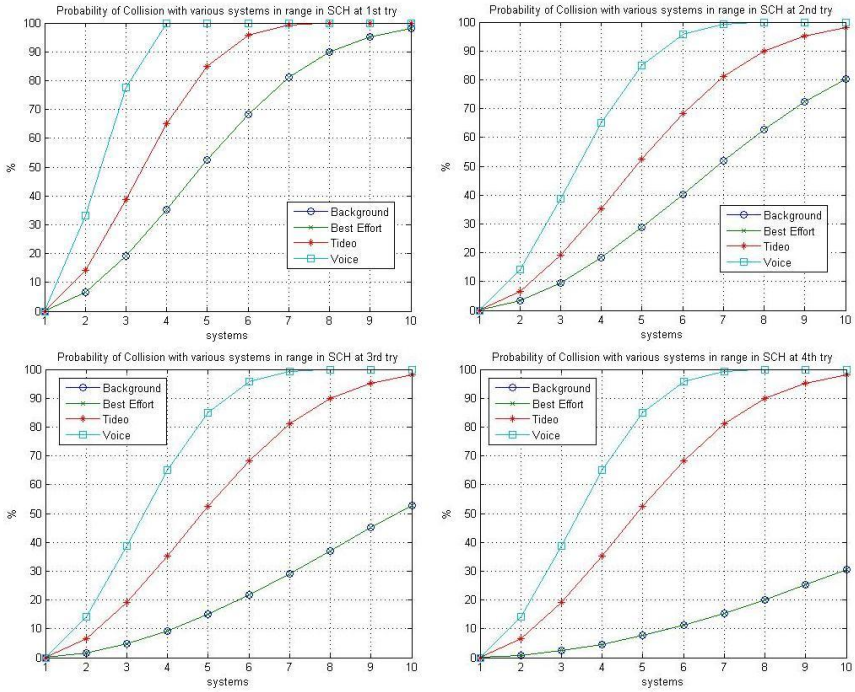


Figure (4.2), Probability of Collision with various systems for 1st, 2nd, 3rd and 4th try for SCH

4.2 Probability of Collision during Retries

The definition of retries which use back off mechanism depends on the new CW value on the systems. For a specific packet with its CW value, if collision occurs, it will increase its CW value or "back-off". From table (2.3) in Section 2.2.4, with the retry limit of seven, the system will follow the value in the table and increase their CW value. As the result, assume only two systems are in the communicable range, the probability of collision is computed as follows in Equation (4.4).

$$\frac{\mathbf{1}}{\mathbf{CW + 1}} \qquad (\mathbf{4.4})$$

With the increasing of CW value, the probability will decrease. This result matches the basic scenario of back-off mechanism which lowers the probability of collision after the system increasing its CW value. Figure (4.3) and figure (4.4) show the probability of collision with two systems and various retries for CCH and SCH. As the number of retries increase, the probability of collision of decrease because of larger CW value. On the other hand, as the number of retries decrease, the probability of collision will increase because smaller CW value.

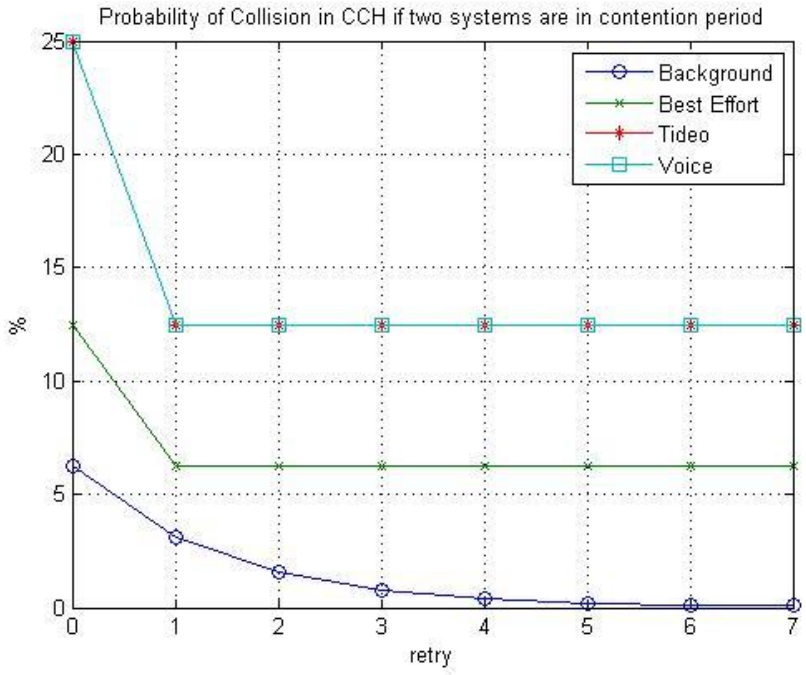


Figure (4.3), Probability of Collision with two systems for various retries for CCH

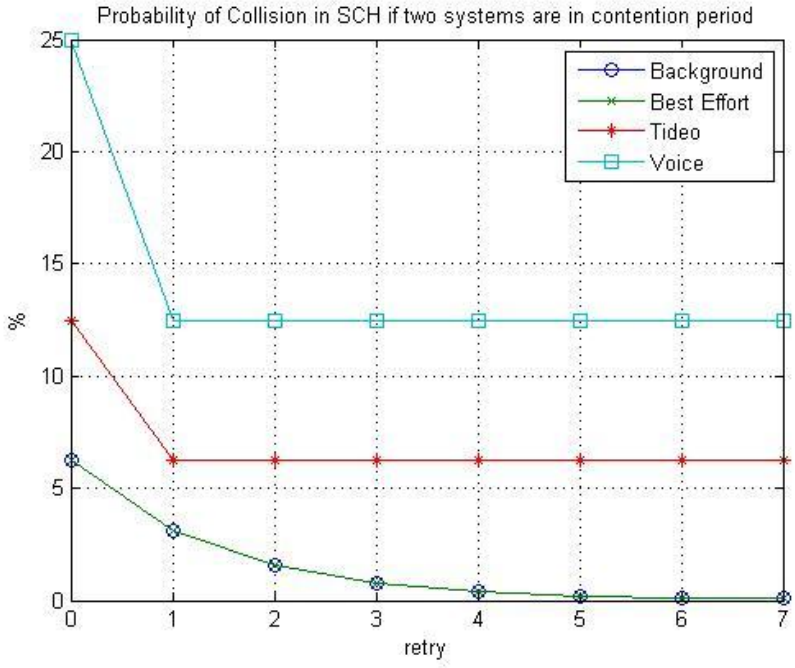


Figure (4.4), Probability of Collision with two systems for various retries for SCH

4.3 Cross Probability with Traffic and Retries

In this Section, a cross comparison of the probability of collision between traffic and retries will be presented. From Section 4.1 and Section 4.2, we already know the effect dependence of the probability depending on different traffic parameters and number of retries. If we connect both two factors together, the result will be like a continuous 3D graph for figure (4.1) and figure (4.2). As the result in figure (4.5), high traffic and low retries will cause the greatest probability of collision because of two major causes happen at a same time. On the other hand, more retires and fewer systems will cause the probability of collision staying low.

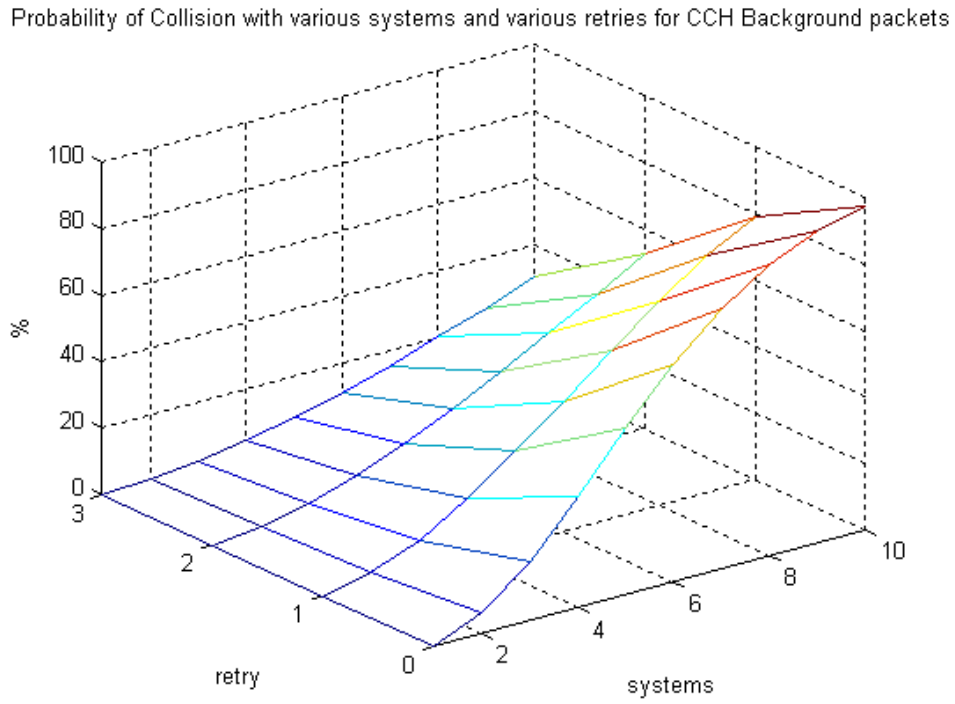


Figure (4.5), Probability of Collision with various systems and retries for background packets in CCH

4.4 Function Validation with Cases

In this section, the actual code will be tested into the previous four cases. They are: Case (1): Channel is idle and AIFS+CW is not zero, Case (2): Channel is busy and AIFS+CW is not zero, Case (3): Internal Collision and AIFS+CW is zero and Case (4): No Collision and AIFS+CW is zero. CCH is the channel to be tested. The simulation result is in Appendix D.

To begin, all the queues in CCH load with two packets. After the packets are loaded into queues, AISFN+CW value will be assigned (line 1-4). At this point, the system is ready to start simulation. The system will have to count a slot time and then decrement the counters until one or more than one counter become zero without interception (line 6-7). At line 8, priority 3 packet is ready to be sent since its corresponding counter is becoming zero. Before dequeuing the packet from priority 3 queue, the system has to make sure the packet has been sent successfully. Here, a function called “peek” will duplicate the packet and transfer to physical layer (line 9). Also, after we peek the packet, we have to do contention window adjustment (line 10-11). Finally, the packet is successfully sent, and then we dequeue the packet and assign new AIFSN_CW value if there is any other packet in the queue (line 12). At this point, case (1) and case (4) have been tested.

Case (2) represents the case that the medium has occupied by other system. In the test, we generate a testing signal which will assert by the chance of 3%. In the real system, case (2) will be processed in carrier sensed thread which detects interruptions. The earliest interruption occurs at line 27. All the counters with some packets in its queue will

have to adjust. As the result, line 28-29 presents the adjusted contention window values and waits for next contention period.

Case (3) represents the case that more than two counter are zero. According to previous section, the scenario is called “internal collision”. The earliest internal collision occurs at line 18-20. At this point, both priority 3 and priority 2 are ready, and the system will choose priority 3 because of basic priority level. As the result, packet in priority 3 queue is being sent to physical layer for transmission. Other queues other than priority 3 queue will be reset.

The entire test script shows the correct scenario according the contention period section.

4.5 Average Transmission Time from Actual Measurement

By measuring and gathering the random transmission time from actual system in CCH for 30 trials for safety messages generated at 10Hz, we made the plot for Safety Message Tx to Rx time with random priorities in figure (4.6). In figure (4.6), the random generated transmission time due to different contention windows is shown different packets, where priority 0 is Background, priority 1 is Best Effort, priority 2 is Video and priority 3 is Voice. By observation, the random transmission time for low priority packets such as Background and Best Effort will almost be guaranteed shorter than high priority packets such as Video and Voice. Finally, the calculated average transmission time will be 0.8514 seconds, 0.1714 seconds, 0.0906 seconds and 0.0655 seconds for the first transmission (see appendix E).

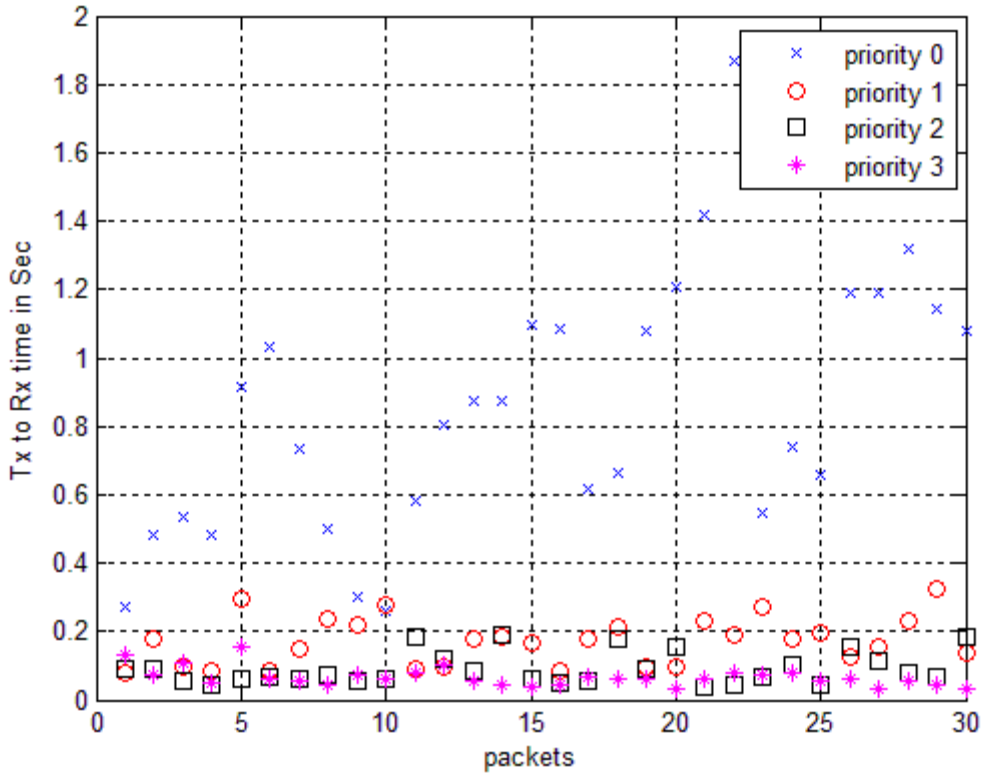


Figure (4.6), Safety Message Tx to Rx time with random priorities

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

Communication among Automobiles will one day play an important role in car safety. In this thesis, the delivery of packets by their priority level is presented by the solution of AIFS and CW mechanism. The algorithm is a part in MAC layer, which controls data accesses issue, including NAV and collision.

In chapter 4, a series of testing for probability of collision in high-low traffic and retries and combination of high-low traffic and retries proves that the protocol has the ability to solve and control in different scenarios, especially, lower the probability of collision for next transmission after collisions occurred.

To conclude, although the system has not been fully developed, there are potential that this system will be efficient and useful in the future.

5.2 Future Work

As mentioned in Section 1.1, collision presents itself as the most serious problem for the entire design. Currently, our system cannot determine whether the packet is lost or collided. By conventional, a packet is missing, the system will retry the packet within retry limits. In the event of collision, the system has to enter back-off process and retry. Since we cannot tell the differences, a suggested combined solution in WAVE could be

start the contention period after adjusting CW values. The advantage of this solution could be not wasting too much time on the retry for a specific packet if there are some higher priorities packets are waiting on the deck. In the future, collision solving could still be a very interesting topic.

BIBLIOGRAPHY

[1] ”IEEE Trial-Use Standard for Wireless Access in Vehicular Environments (WAVE) - Multi-channel Operation,” IEEE Std 1609.4-2006 , vol., no., pp.c1-74, 2006

[2] ”IEEE standard for information technology- telecommunications and information exchange between systems- local and metropolitan area networks- specific requirements Part II: wireless LAN medium access control (MAC) and physical layer (PHY) specifications,“ IEEE Std 802.11g-2003 (Amendment to IEEE Std 802.11, 1999 Edn. (Reaff 2003) as amended by IEEE Stds 802.11a-1999, 802.11b-1999, 802.11b-1999/Cor 1-2001, and 802.11d-2001) , vol., no., pp.i-67, 2003

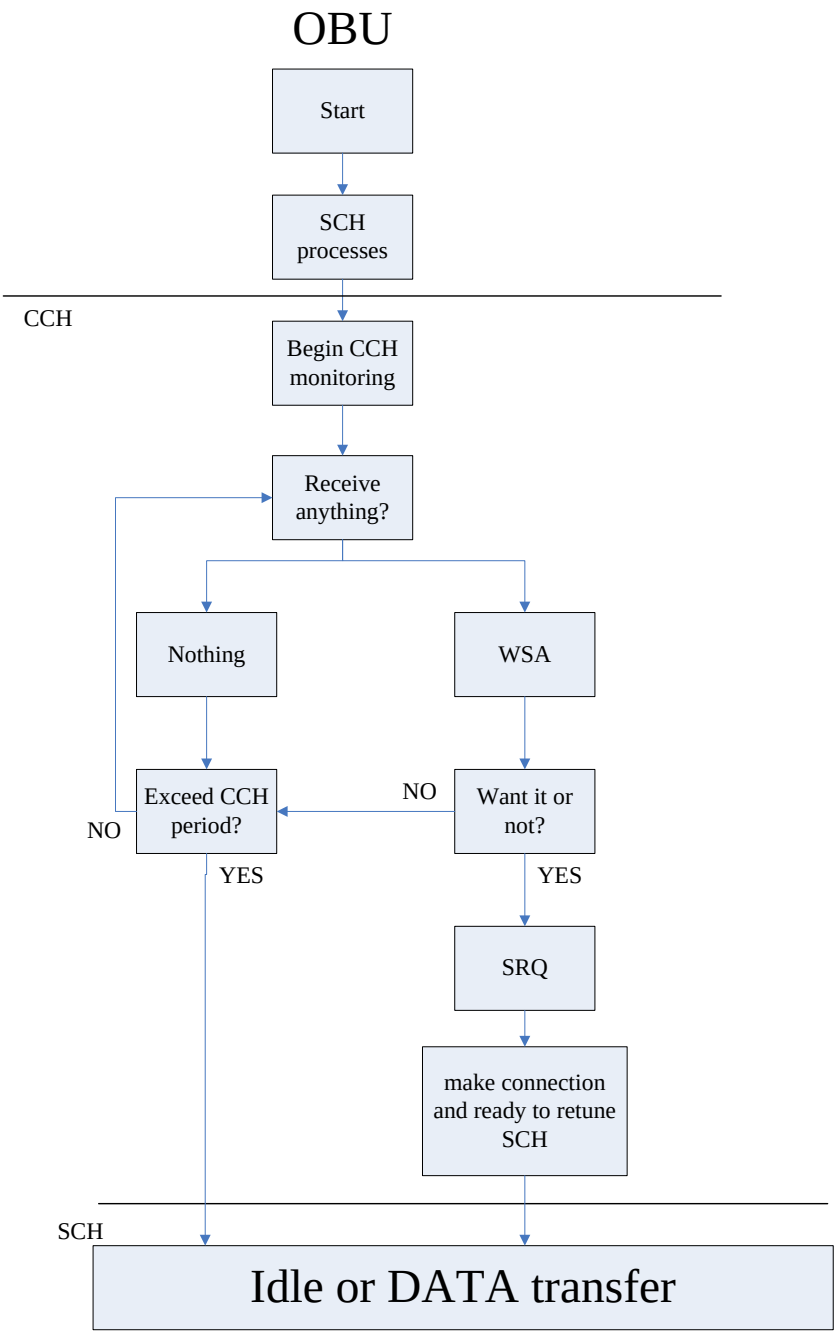
[3] IEEE Computer Society, IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, LAN/MAN Standards Committee, 2007

[4] IEEE P802.11p/D3.05, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY), March 2008

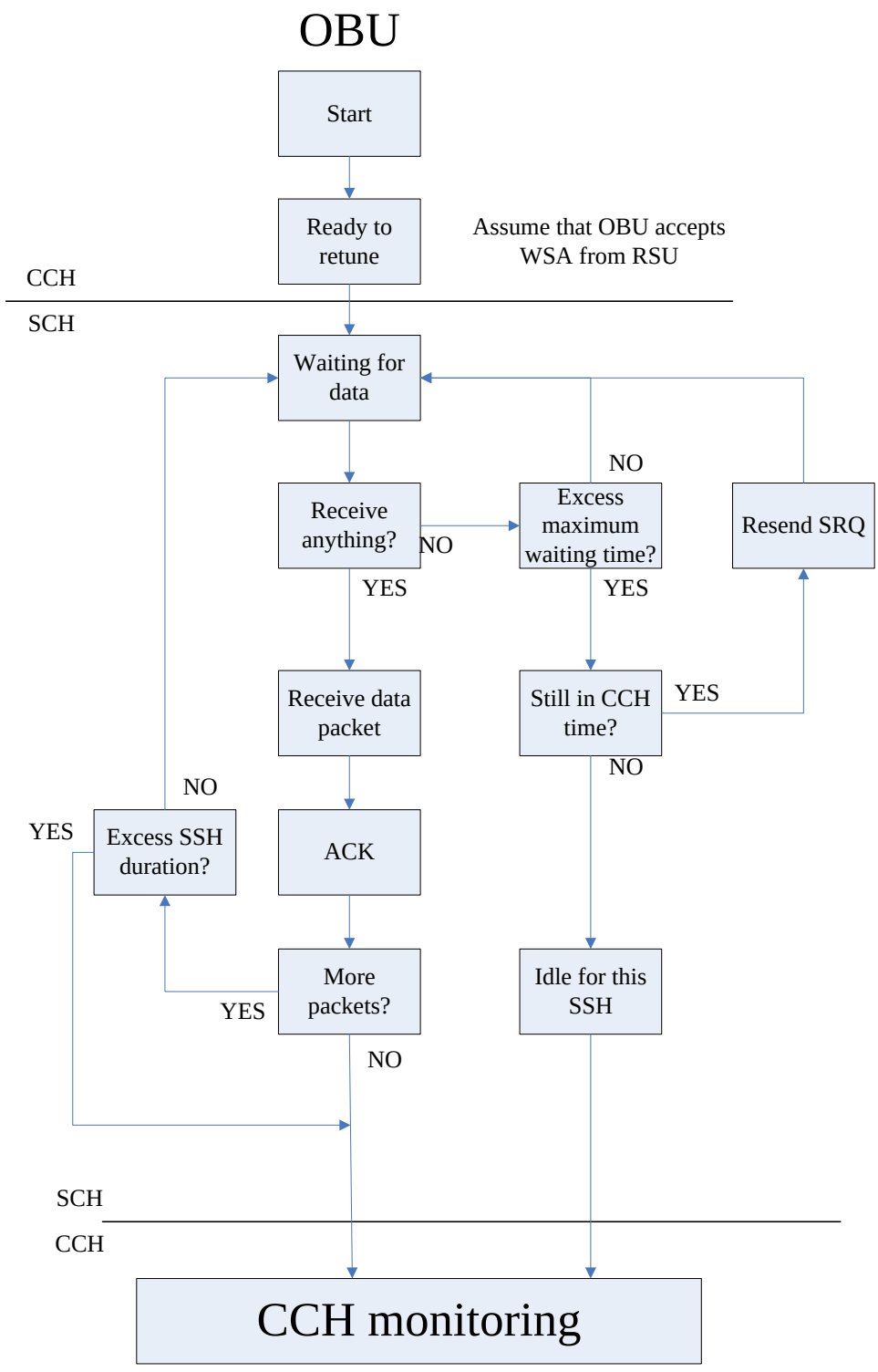
[5] Matthew S. Gast, Wireless Networks: The Definitive Guide, 2e, O’Reilly, 2005

APPENDIX

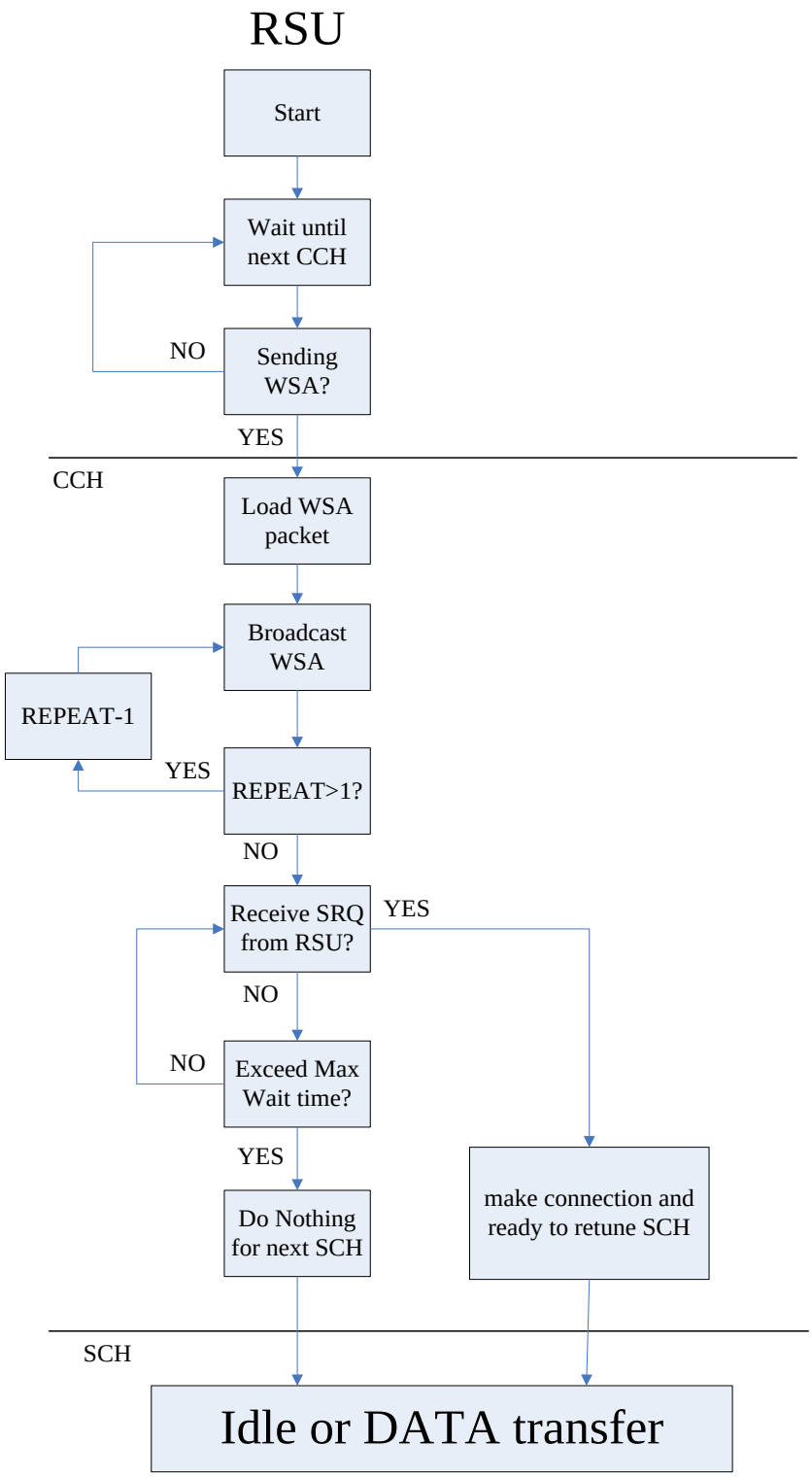
A. Flowcharts



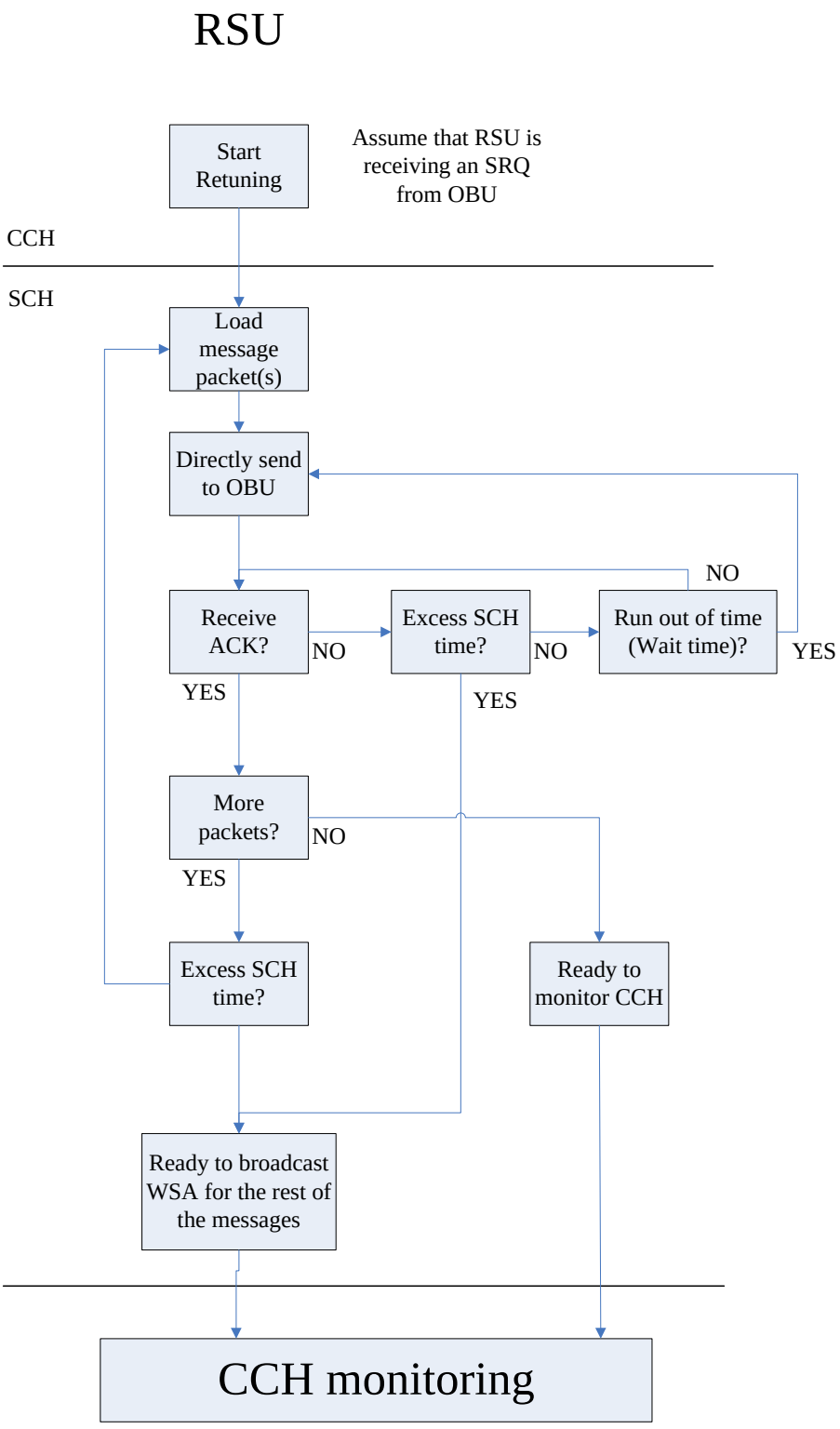
Flowchart 1, flowchart for OBU in CCH period



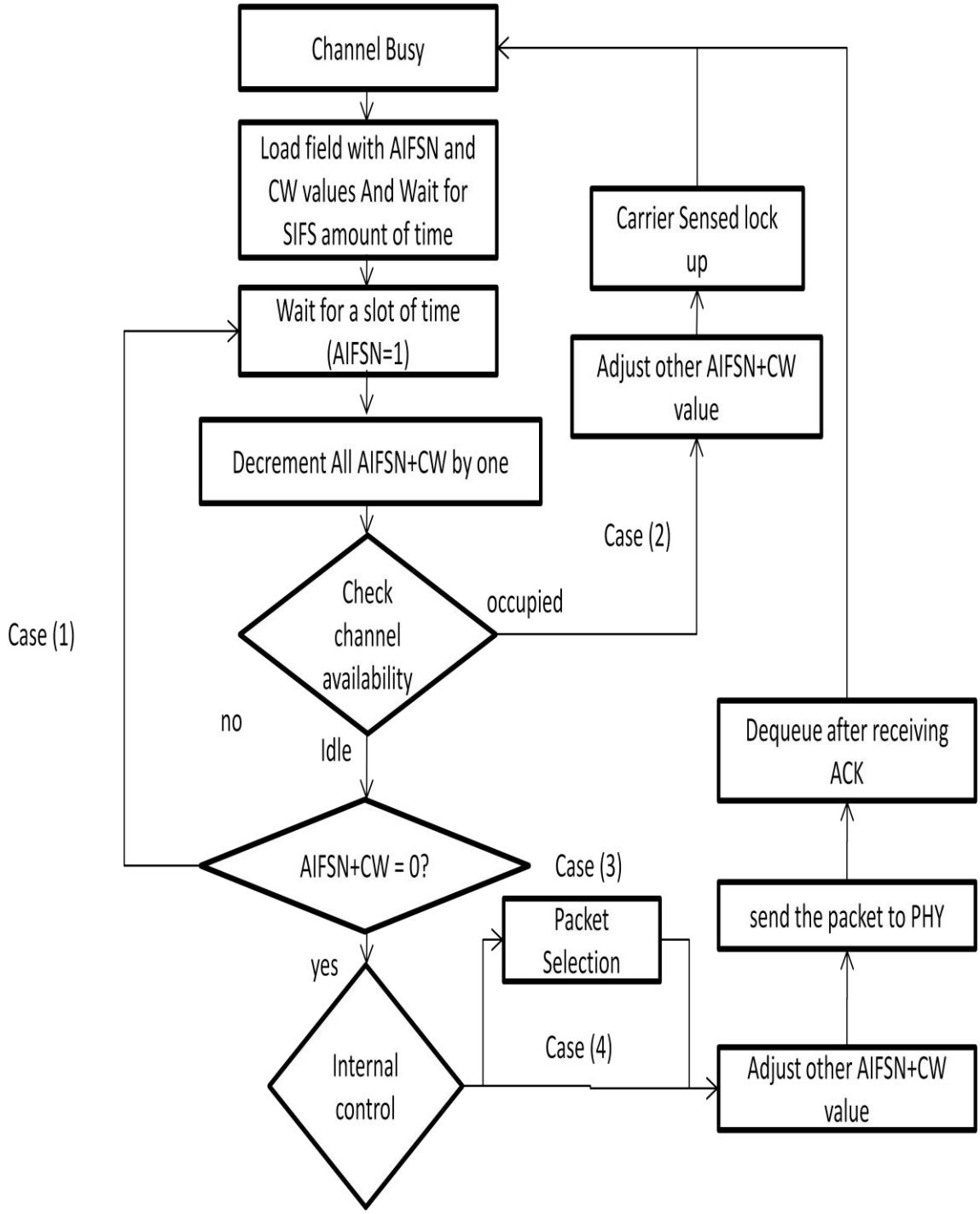
Flowchart 2, flowchart for OBU in SCH period



Flowchart 3, flowchart for RSU in CCH period



Flowchart 4, flowchart for RSU in SCH period



Flowchart 5, Procedure flowchart for AIFSN/CW counters with internal and external control involved

B. Actual Function Code

```
import random

class Cell_In_Queue:
#use in queue with pointers
    def __init__(self,packet):
        self.packet=packet
        self.next=0

class Queue:
#queue itself with three fields: start, end and length
    def __init__(self):
        self.start=None
        self.end=None
        self.length=0

    def Get_In_Queue(self,packet):
    #push into queue
        new_packet=Cell_In_Queue(packet)
        if self.length>0:
            self.end.next=new_packet
            self.end=new_packet
        else:
            self.start=new_packet
            self.end=new_packet
        self.length=self.length+1

    def Get_Out_Queue(self):
    #pop from queue
        out_packet=self.start.packet
        if self.length==1:
            self.start=None
            self.end=None
        else:
            self.start=self.start.next
        self.length=self.length-1
        return out_packet


class Four_Priority_Queue:
    def __init__(self,AISFN,CW_range):
        self.fourQueues=[Queue(), Queue(),Queue(),Queue()]
        self.fourAIFSN_CW=[0,0,0,0]
        self.fourAIFSN=AISFN
        self.fourCW_range=CW_range
```

```
        self.counter=0

    def Any_Packet(self):
    # return true if four queues are not completely empty
        for i in range(0,4):
            if self.fourQueues[i].length==0:
                return 0
        return 1

    def Adjust_Single_CW(self,queue):
    # Adjust single AIFSN+CW value
        if self.fourQueues[queue].length!=0:
            if self.counter>self.fourAIFSN[queue]:

self.fourAIFSN_CW[queue]=self.fourAIFSN_CW[queue]+self.fourAIFSN[queue]
            else:
                self.fourAIFSN_CW[queue]=self.fourAIFSN_CW[queue]+self.counter


    def Adjust_Four_CW(self,number):
    # Adjust four AIFSN+CW value
        if number!=0 and self.fourQueues[0].length!=0:
            self.Adjust_Single_CW(0)
        if number!=1 and self.fourQueues[1].length!=0:
            self.Adjust_Single_CW(1)
        if number!=2 and self.fourQueues[2].length!=0:
            self.Adjust_Single_CW(2)
        if number!=3 and self.fourQueues[3].length!=0:
            self.Adjust_Single_CW(3)


class Queues_In_Three_Channel:
    def __init__(self):
        self.Channel_SCH=Four_Priority_Queue([7,3,2,2],[15,15,7,3])
        self.Channel_CCH=Four_Priority_Queue([9,6,3,2],[15,7,3,3])
        self.Channel_C2C=Four_Priority_Queue([9,6,3,2],[15,7,3,3])

    def Any_Packet_In_Channel(self,channel):
    # return true if there is any packet in the channel
        if channel=="SCH":
            return self.Channel_SCH.Any_Packet()
        elif channel=="CCH":
            return self.Channel_CCH.Any_Packet()
        elif channel=="C2C":
            return self.Channel_C2C.Any_Packet()
        else:
```

```
            return -1

    def Dequeue(self,channel,priority,payload):
    #returns the packet in queue_number and channel number
        if channel=="SCH":
            if self.Channel_SCH.fourQueues[priority].length>1:

self.Channel_SCH.fourAIFSN_CW[priority]=self.Channel_SCH.fourAIFSN[priority]+ra
ndom.randint(0,self.Channel_SCH.fourCW_range[priority])
            else:
                self.Channel_SCH.fourAIFSN_CW[priority]=0
            return self.Channel_SCH.fourQueues[priority].Get_Out_Queue()
        elif channel=="CCH":
            if self.Channel_CCH.fourQueues[priority].length>1:

self.Channel_CCH.fourAIFSN_CW[priority]=self.Channel_CCH.fourAIFSN[priority]+r
andom.randint(0,self.Channel_CCH.fourCW_range[priority])
            else:
                self.Channel_CCH.fourAIFSN_CW[priority]=0
            return self.Channel_CCH.fourQueues[priority].Get_Out_Queue()
        elif channel=="C2C":
            if self.Channel_C2C.fourQueues[priority].length>1:

self.Channel_C2C.fourAIFSN_CW[priority]=self.Channel_C2C.fourAIFSN[priority]+ra
ndom.randint(0,self.Channel_C2C.fourCW_range[priority])
            else:
                self.Channel_C2C.fourAIFSN_CW[priority]=0
            return self.Channel_C2C.fourQueues[priority].Get_Out_Queue()
        else:
            return -1

    def Enqueue(self,queue_number,channel,packet):
    #insert the packet into queue_number in channel
        #print "Enqueue channel = %s priority = %d length queue = %d, CW %d" %
( channel, queue_number, self.Channel_C2C.fourQueues[queue_number].length,
self.Channel_C2C.fourAIFSN_CW[queue_number]  )
        if channel=="SCH":
            if self.Channel_SCH.fourQueues[queue_number].length==0:

self.Channel_SCH.fourAIFSN_CW[queue_number]=self.Channel_SCH.fourAIFSN[que
ue_number]+random.randint(0,self.Channel_SCH.fourCW_range[queue_number])
            self.Channel_SCH.fourQueues[queue_number].Get_In_Queue(packet)
        elif channel=="CCH":
            if self.Channel_CCH.fourQueues[queue_number].length==0:
```

```
self.Channel_CCH.fourAIFSN_CW[queue_number]=self.Channel_CCH.fourAIFSN[que
ue_number]+random.randint(0,self.Channel_CCH.fourCW_range[queue_number])
            self.Channel_CCH.fourQueues[queue_number].Get_In_Queue(packet)
        elif channel=="C2C":
            if self.Channel_C2C.fourQueues[queue_number].length==0:

self.Channel_C2C.fourAIFSN_CW[queue_number]=self.Channel_C2C.fourAIFSN[queu
e_number]+random.randint(0,self.Channel_C2C.fourCW_range[queue_number])
            self.Channel_C2C.fourQueues[queue_number].Get_In_Queue(packet)
        else:
            return -1

    def Is_Packet_Ready(self,channel):
    #check if counter value matches any CW value on the channel
        #print "Checking channel %s, p2 has %d length p2 %d" %(channel,
self.Channel_C2C.fourAIFSN_CW[2], self.Channel_C2C.fourQueues[2].length )
        if channel=="SCH":
            for i in range(0,4):
                if self.Channel_SCH.fourAIFSN_CW[i]<=0 and
self.Channel_SCH.fourQueues[i].length>0:
                    return 1
            return 0
        elif channel=="CCH":
            for i in range(0,4):
                if self.Channel_CCH.fourAIFSN_CW[i]<=0 and
self.Channel_CCH.fourQueues[i].length>0:
                    return 1
            return 0
        elif channel=="C2C":
            for i in range(0,4):
                if self.Channel_C2C.fourAIFSN_CW[i]<=0 and
self.Channel_C2C.fourQueues[i].length>0:
                    return 1
            return 0
        else:
            return -1


    def Peek(self,channel):
    #return the smallest CW value or the highest priority and adjust AIFSN+CW value
        queue=1
        payload=None
        #print "Peeking at channel %s" % channel
        if channel=="SCH":
            for i in range(0,4):
```

```
            if self.Channel_SCH.fourAIFSN_CW[i]<=0 and
self.Channel_SCH.fourQueues[i].length>0:
                payload=self.Channel_SCH.fourQueues[i].start.packet
                queue=i
          self.Channel_SCH.Adjust_Four_CW(queue)
          self.Channel_SCH.counter=0
       elif channel=="CCH":
          for i in range(0,4):
            if self.Channel_CCH.fourAIFSN_CW[i]<=0 and
self.Channel_CCH.fourQueues[i].length>0:
                payload=self.Channel_CCH.fourQueues[i].start.packet
                queue=i
          self.Channel_CCH.Adjust_Four_CW(queue)
          self.Channel_CCH.counter=0
       elif channel=="C2C":
          for i in range(0,4):
            if self.Channel_C2C.fourAIFSN_CW[i]<=0 and
self.Channel_C2C.fourQueues[i].length>0:
                payload=self.Channel_C2C.fourQueues[i].start.packet
                queue=i
          self.Channel_C2C.Adjust_Four_CW(queue)
          self.Channel_C2C.counter=0
       #print "Peeking at channel %s, queue = %d type payload = %s, packet %s" %
(channel, queue, type( payload ),
type( self.Channel_C2C.fourQueues[queue].start.packet ) )

       return { 'payload':payload, 'priority':queue }

    def Decrement(self,Channel):
    #decrement CW value on Channel
       #print "Decrement channel = %s, counter = %d, CW = %d" %( Channel,
self.Channel_C2C.counter, self.Channel_C2C.fourAIFSN_CW[2] )
       if Channel=="SCH":
          for i in range(0,4):
            self.Channel_SCH.fourAIFSN_CW[i]=self.Channel_SCH.fourAIFSN_CW[i]-
1
            if self.Channel_SCH.fourAIFSN_CW[i]<0:
               self.Channel_SCH.fourAIFSN_CW[i] = 0
          self.Channel_SCH.counter=self.Channel_SCH.counter+1
       elif Channel=="CCH":
          for i in range(0,4):
            self.Channel_CCH.fourAIFSN_CW[i]=self.Channel_CCH.fourAIFSN_CW[i]-
1
            if self.Channel_CCH.fourAIFSN_CW[i]<0:
               self.Channel_CCH.fourAIFSN_CW[i] = 0
          self.Channel_CCH.counter=self.Channel_CCH.counter+1
```

```
        elif Channel=="C2C":
            for i in range(0,4):
                self.Channel_C2C.fourAIFSN_CW[i]=self.Channel_C2C.fourAIFSN_CW[i]-1
                if self.Channel_C2C.fourAIFSN_CW[i]<0:
                    self.Channel_C2C.fourAIFSN_CW[i] = 0
            self.Channel_C2C.counter=self.Channel_C2C.counter+1
        else:
            return -1

    def Ini_CW_Adjust(self,channel):
    #(interruption) set counter back to zero and adjust CW for all queues
        #print "Reseting channel %s" % channel
        if channel=="SCH":
            for i in range(0,4):
                self.Channel_SCH.Adjust_Single_CW(i)
            self.Channel_SCH.counter=0
        elif channel=="CCH":
            for i in range(0,4):
                self.Channel_CCH.Adjust_Single_CW(i)
            self.Channel_CCH.counter=0
        elif channel=="C2C":
            for i in range(0,4):
                self.Channel_C2C.Adjust_Single_CW(i)
            self.Channel_C2C.counter=0
        else:
            return -1

    def Print_AIFSN_CW(self,Channel):
    # print out AIFSN+CW value (testing use)
        if Channel=="SCH":
            print "AIFSN+CW value (%i %i %i %i)"
%(queues.Channel_SCH.fourAIFSN_CW[0],queues.Channel_SCH.fourAIFSN_CW[1],q
ueues.Channel_SCH.fourAIFSN_CW[2],queues.Channel_SCH.fourAIFSN_CW[3])
        elif Channel=="CCH":
            print "AIFSN+CW value (%i %i %i %i)"
%(queues.Channel_CCH.fourAIFSN_CW[0],queues.Channel_CCH.fourAIFSN_CW[1],
queues.Channel_CCH.fourAIFSN_CW[2],queues.Channel_CCH.fourAIFSN_CW[3])
        elif Channel=="C2C":
            print "AIFSN+CW value (%i %i %i %i)"
%(queues.Channel_C2C.fourAIFSN_CW[0],queues.Channel_C2C.fourAIFSN_CW[1],q
ueues.Channel_C2C.fourAIFSN_CW[2],queues.Channel_C2C.fourAIFSN_CW[3])
```

C. Actual Testing Code

```
#testing loop
queues=Queues_In_Three_Channel()
queues.Enqueue(0,"CCH","packet0_1")
queues.Enqueue(1,"CCH","packet1_1")
queues.Enqueue(2,"CCH","packet2_1")
queues.Enqueue(3,"CCH","packet3_1")
print "Enqueue first packet into queues"
queues.Print_AIFSN_CW("CCH")
queues.Enqueue(0,"CCH","packet0_2")
queues.Enqueue(1,"CCH","packet1_2")
queues.Enqueue(2,"CCH","packet2_2")
queues.Enqueue(3,"CCH","packet3_2")
print "Enqueue second packet into queues"
queues.Print_AIFSN_CW("CCH")
print "Start processing simulation"
for i in range (1,9):
   queues.Decrement("CCH")
   queues.Print_AIFSN_CW("CCH")
   while not queues.Is_Packet_Ready("CCH"):
      if random.randint(1,30)>29:
         print "Other system obtain the medium"
         print "reset AIFSN+CW"
         queues.Ini_CW_Adjust("CCH")
         queues.Print_AIFSN_CW("CCH")
         print "restart at next contention period"
      else:
         queues.Decrement("CCH")
         queues.Print_AIFSN_CW("CCH")
   print "packet to be sent to PHY"
   result=queues.Peek("CCH")
   print result
   print "adjust other queues"
   queues.Print_AIFSN_CW("CCH")
   print "Dequeue after transmission"
   temp=queues.Dequeue("CCH",result['priority'],result['payload'])
   queues.Print_AIFSN_CW("CCH")
print "End of Processes"
```

D. Simulation Results

Line	Action
1	Enqueue first packet into queues
2	AIFSN+CW value (11 13 5 2)
3	Enqueue second packet into queues
4	AIFSN+CW value (11 13 5 2)
5	Start processing simulation
6	AIFSN+CW value (10 12 4 1)
7	AIFSN+CW value (9 11 3 0)
8	packet to be sent to PHY
9	{'priority': 3, 'payload': 'packet3_1'}
10	adjust other queues
11	AIFSN+CW value (11 13 5 0)
12	Dequeue after transmission
13	AIFSN+CW value (11 13 5 5)
14	AIFSN+CW value (10 12 4 4)
15	AIFSN+CW value (9 11 3 3)
16	AIFSN+CW value (8 10 2 2)
17	AIFSN+CW value (7 9 1 1)
18	AIFSN+CW value (6 8 0 0)
19	packet to be sent to PHY
20	{'priority': 3, 'payload': 'packet3_2'}
21	adjust other queues
22	AIFSN+CW value (11 13 3 0)
23	Dequeue after transmission
24	AIFSN+CW value (11 13 3 0)
25	AIFSN+CW value (10 12 2 0)
26	AIFSN+CW value (9 11 1 0)
27	Other system obtain the medium
28	reset AIFSN+CW
29	AIFSN+CW value (11 13 3 0)
30	restart at next contention period
31	AIFSN+CW value (10 12 2 0)
32	AIFSN+CW value (9 11 1 0)
33	AIFSN+CW value (8 10 0 0)
34	packet to be sent to PHY
35	{'priority': 2, 'payload': 'packet2_1'}
36	adjust other queues

```
    AIFSN+CW value (11 13 0 0)
    Dequeue after transmission
37  AIFSN+CW value (11 13 5 0)
38  AIFSN+CW value (10 12 4 0)
39  AIFSN+CW value (9 11 3 0)
40  AIFSN+CW value (8 10 2 0)
41  AIFSN+CW value (7 9 1 0)
42  AIFSN+CW value (6 8 0 0)
43  packet to be sent to PHY
44  {'priority': 2, 'payload': 'packet2_2'}
45  adjust other queues
46  AIFSN+CW value (11 13 0 0)
47  Dequeue after transmission
48  AIFSN+CW value (11 13 0 0)
49  AIFSN+CW value (10 12 0 0)
50  AIFSN+CW value (9 11 0 0)
51  AIFSN+CW value (8 10 0 0)
52  AIFSN+CW value (7 9 0 0)
53  AIFSN+CW value (6 8 0 0)
54  AIFSN+CW value (5 7 0 0)
55  AIFSN+CW value (4 6 0 0)
56  AIFSN+CW value (3 5 0 0)
57  AIFSN+CW value (2 4 0 0)
58  AIFSN+CW value (1 3 0 0)
59  AIFSN+CW value (0 2 0 0)
60  packet to be sent to PHY
61  {'priority': 0, 'payload': 'packet0_1'}
62  adjust other queues
63  AIFSN+CW value (0 8 0 0)
64  Dequeue after transmission
65  AIFSN+CW value (13 8 0 0)
66  AIFSN+CW value (12 7 0 0)
67  AIFSN+CW value (11 6 0 0)
68  AIFSN+CW value (10 5 0 0)
69  AIFSN+CW value (9 4 0 0)
70  AIFSN+CW value (8 3 0 0)
71  Other system obtain the medium
72  reset AIFSN+CW
73  AIFSN+CW value (13 8 0 0)
```

restart at next contention period
AIFSN+CW value (12 7 0 0)
AIFSN+CW value (11 6 0 0)
AIFSN+CW value (10 5 0 0)
Other system obtain the medium
reset AIFSN+CW
AIFSN+CW value (13 8 0 0)
restart at next contention period
AIFSN+CW value (12 7 0 0)
AIFSN+CW value (11 6 0 0)
AIFSN+CW value (10 5 0 0)
AIFSN+CW value (9 4 0 0)
AIFSN+CW value (8 3 0 0)
AIFSN+CW value (7 2 0 0)
AIFSN+CW value (6 1 0 0)
AIFSN+CW value (5 0 0 0)
packet to be sent to PHY
{'priority': 1, 'payload': 'packet1_1'}
adjust other queues
AIFSN+CW value (13 0 0 0)
Dequeue after transmission
AIFSN+CW value (13 10 0 0)
AIFSN+CW value (12 9 0 0)
AIFSN+CW value (11 8 0 0)
AIFSN+CW value (10 7 0 0)
Other system obtain the medium
reset AIFSN+CW
AIFSN+CW value (13 10 0 0)
restart at next contention period
AIFSN+CW value (12 9 0 0)
AIFSN+CW value (11 8 0 0)
Other system obtain the medium
reset AIFSN+CW
AIFSN+CW value (13 10 0 0)
restart at next contention period
AIFSN+CW value (12 9 0 0)
AIFSN+CW value (11 8 0 0)
AIFSN+CW value (10 7 0 0)
AIFSN+CW value (9 6 0 0)

```
AIFSN+CW value (8 5 0 0)
AIFSN+CW value (7 4 0 0)
AIFSN+CW value (6 3 0 0)
AIFSN+CW value (5 2 0 0)
AIFSN+CW value (4 1 0 0)
AIFSN+CW value (3 0 0 0)
packet to be sent to PHY
{'priority': 1, 'payload': 'packet1_2'}
adjust other queues
AIFSN+CW value (12 0 0 0)
Dequeue after transmission
AIFSN+CW value (12 0 0 0)
AIFSN+CW value (11 0 0 0)
AIFSN+CW value (10 0 0 0)
AIFSN+CW value (9 0 0 0)
AIFSN+CW value (8 0 0 0)
AIFSN+CW value (7 0 0 0)
AIFSN+CW value (6 0 0 0)
AIFSN+CW value (5 0 0 0)
AIFSN+CW value (4 0 0 0)
AIFSN+CW value (3 0 0 0)
AIFSN+CW value (2 0 0 0)
AIFSN+CW value (1 0 0 0)
AIFSN+CW value (0 0 0 0)
packet to be sent to PHY
{'priority': 0, 'payload': 'packet0_2'}
adjust other queues
AIFSN+CW value (0 0 0 0)
Dequeue after transmission
AIFSN+CW value (0 0 0 0)
End of Processes
```

E. Actual Transmission Data

	Priority 0	Priority 1	Priority 2	Priority 3
1	0.271545	0.081072	0.091344	0.133194
2	0.482998	0.180674	0.088158	0.071460
3	0.532301	0.097053	0.055315	0.108780
4	0.480281	0.082002	0.043323	0.047214
5	0.913133	0.297785	0.064125	0.156611
6	1.029454	0.086482	0.065108	0.062931
7	0.735972	0.148378	0.064033	0.056907
8	0.498674	0.239354	0.075207	0.045522
9	0.300902	0.217505	0.056837	0.070831
10	0.258600	0.277410	0.058798	0.058799
11	0.581074	0.093321	0.182839	0.078026
12	0.806979	0.097172	0.117085	0.099858
13	0.872027	0.179611	0.086517	0.053317
14	0.876244	0.184047	0.189126	0.042783
15	1.098761	0.164730	0.061075	0.039486
16	1.086406	0.086500	0.048167	0.046370
17	0.615979	0.179723	0.058017	0.067478
18	0.664509	0.214191	0.180878	0.064004
19	1.077021	0.097103	0.088177	0.059436
20	1.207611	0.098991	0.152198	0.034889
21	1.417400	0.228277	0.038140	0.062861
22	1.867762	0.191223	0.044693	0.081458
23	0.548315	0.272067	0.064432	0.072002
24	0.737526	0.179168	0.099682	0.077534
25	0.656268	0.197336	0.044505	0.054226
26	1.189228	0.125914	0.154296	0.059102
27	1.188930	0.153018	0.113892	0.029839
28	1.320855	0.229819	0.079353	0.054247
29	1.145132	0.322994	0.064914	0.041743
30	1.080913	0.137906	0.186739	0.032766
Average	**0.851427**	**0.171361**	**0.090566**	**0.065456**

Unit: Seconds